

PATENT APPLICATION

PROCESS VERIFICATION

Inventor:

Warner Cockerille
414 Abbay Way
Sparks, NV 89431
U.S. Citizen

Steven G. LeMay
17085 Castle Pine Dr.
Reno, NV 89511
U.S. Citizen

Robert Breckner
4519 Eagle Mountain Dr.
Sparks, NV 89431
U.S. Citizen

Assignee:

IGT

BEYER WEAVER & THOMAS, LLP
P.O. Box 778
Berkeley, CA 94704
Telephone (510) 843-6200

PROCESS VERIFICATION

RELATED APPLICATION DATA

5 The present application is a continuation of and claims priority under U.S.C. 120 from co-pending U.S. Patent Application No. 09/925,098, entitled "PROCESS VERIFICATION" filed on August 8, 2001, which is incorporated herein by reference in its entirety for all purposes.

BACKGROUND OF THE INVENTION

10 This invention relates to gaming machines such as video slot machines and video poker machines. More particularly, the present invention relates to methods of verifying the authenticity of gaming software executed on a gaming machine.

15 Typically, utilizing a master gaming controller, a gaming machine controls various combinations of devices that allow a player to play a game on the gaming machine and also encourage game play on the gaming machine. For example, a game played on a gaming machine usually requires a player to input money or indicia of credit into the gaming machine, indicate a wager amount, and initiate a game play. These steps require the gaming machine to control input devices, including bill validators and coin acceptors, to accept money into the gaming machine and
20 recognize user inputs from devices, including touch screens and button pads, to determine the wager amount and initiate game play. After game play has been initiated, the gaming machine determines a game outcome, presents the game outcome to the player and may dispense an award of some type depending on the outcome of the game.

25 As technology in the gaming industry progresses, the traditional mechanically driven reel slot machines are being replaced with electronic counterparts having CRT, LCD video displays or the like and gaming machines such as video slot machines and video poker machines are becoming increasingly popular. Part of the reason for their increased popularity is the nearly endless variety of games that can be
30 implemented on gaming machines utilizing advanced electronic technology. In some

cases, newer gaming machines are utilizing computing architectures developed for personal computers. These video/electronic gaming advancements enable the operation of more complex games, which would not otherwise be possible on mechanical-driven gaming machines and allow the capabilities of the gaming machine to evolve with advances in the personal computing industry.

To implement the gaming features described above on a gaming machine using computing architectures utilized in the personal computer industry, a number of requirements unique to the gaming industry must be considered. One such requirement is the regulation of gaming software. Typically, within a geographic area allowing gaming, i.e. a gaming jurisdiction, a governing entity is chartered with regulating the games played in the gaming jurisdiction to insure fairness and to prevent cheating. Thus, in many gaming jurisdictions, there are stringent regulatory restrictions for gaming machines requiring a time consuming approval process of new gaming software and any software modifications to gaming software used on a gaming machine.

In the past, to implement the play of a game on a gaming machine, a monolithic software architecture has been used. In a monolithic software architecture, a single gaming software executable is developed. The single executable may be burnt onto an EPROM and then submitted to various gaming jurisdictions for approval. After the gaming software is approved, a unique signature can be determined for the gaming software stored on the EPROM using a method such as a CRC. Then, when a gaming machine is shipped to a local jurisdiction, the gaming software signature on the EPROM can be compared with an approved gaming software signature prior to installation of the EPROM on the gaming machine. The comparison process is used to ensure that approved gaming software has been installed on the gaming machine.

A disadvantage of a monolithic programming architecture is that a single executable that works for many different applications can be quite large. For instance, gaming rules may vary from jurisdiction to jurisdiction. Thus, either a single custom executable can be developed for each jurisdiction or one large executable with additional logic can be developed that is valid in many jurisdictions. The customization process may be time consuming and inefficient. For instance, upgrading the gaming software may require developing new executables for each

jurisdiction, submitting the executables for reapproval, and then replacing or reprogramming EPROMs in each gaming machine.

Typically, personal computers use an object oriented software architecture where different software objects may be dynamically linked together prior to execution or even during execution to create many different combinations of executables that perform different functions. Thus, for example, to account for differences in gaming rules between different gaming jurisdictions, gaming software objects appropriate to a particular gaming jurisdiction may be linked at run-time which is simpler than creating a single different executable for each jurisdiction. Also, object oriented software architectures simplify the process of upgrading software since a software object, which usually represents only a small portion of the software, may be upgraded rather than the entire software. However, a disadvantage of object oriented software architectures is that they are not very compatible with EPROMs, which are designed for static executables. Thus, the gaming software regulation process described above using EPROM's may not be applicable to a gaming machine employing an object orientated software approach.

Further, in the past, gaming jurisdictions have required that EPROM based software to "run in place" on the EPROM and not from RAM i.e. the software may not be loaded into RAM for execution. Typically, personal computers load executables from a mass storage device, such as a hard-drive, to RAM and then the software is executed from RAM. Running software from an EPROM limits the size of the executable since the storage available on an EPROM is usually much less than the storage available on a hard-drive. Also, this approach is not generally compatible with PC based devices that load software from a mass storage device to RAM for execution.

In view of the above, methods and apparatus for regulating and verifying gaming software stored in and executed from RAM using object oriented software architectures are needed for gaming machines using these architectures.

SUMMARY OF THE INVENTION

This invention addresses the needs indicated above by providing methods and apparatus for verifying the authenticity of gaming software stored in and executed from RAM on a gaming machine. When presenting a game on the gaming machine, a master gaming controller may dynamically load gaming software applications into RAM and dynamically unload gaming software applications from RAM. The authenticity of the gaming software applications temporarily stored in RAM may be verified by using methods to compare it with certified gaming software stored on one or more local or remote file storage devices accessible to the master gaming controller on the gaming machine. The verification process may be used to satisfy gaming regulatory entities within various gaming jurisdictions that require certified gaming software to be operating on the gaming machine at all times as well as to prevent tampering with the gaming machine.

One aspect of the present invention provides a method of verifying the authenticity of a first gaming software program temporarily stored in RAM of a gaming machine having a master gaming controller for executing the gaming software program. The method may be generally characterized as including: (a) identifying the first gaming software program as currently stored in the gaming machine RAM; (b) identifying a second gaming software program stored on a file storage device; (c) comparing at least a first portion of the second gaming software program with a first portion of the first gaming software program as currently stored in the gaming machine RAM, where the first portion of the gaming software program is a portion of the first gaming software program that does not change during execution of the first gaming software program.

In particular embodiments, the first portion of the first gaming software program may include at least a static header of the first gaming software program or at least executable code of the first gaming software program. The second gaming software program may include a substantially identical copy of the executable code of the first gaming software program. In addition, the second gaming software program may be certified for execution on the gaming machine in one or more gaming jurisdictions by a regulatory entity within each of the gaming jurisdictions. The file storage device may be located on the gaming machine or at a remote location from the gaming machine. The remote file storage device may be a game server.

In yet other embodiments, the method may include one or more of the following: a) generating an error condition when the first portion of the second gaming software program does not match the first portion of the first gaming software program stored in RAM, b) comparing a plurality of portions of the second gaming

software program with a plurality of portions of the first gaming software program as currently stored in the gaming machine RAM, c) generating an error condition when at least one of the plurality of compared portions of the second gaming software program does not match at least one of the plurality of portions of the first gaming software program stored in RAM, d) identifying an executable file name for the first gaming software program, e) identifying the second gaming software program using the executable file name, f) identifying a memory location in RAM of the first gaming software program, g) identifying the first gaming software program from a directory of processes scheduled for execution on the gaming machine, h) selecting the second gaming software program from a list of certified gaming software programs wherein the certified gaming software programs are stored on one or more file storage devices and i) presenting a game of chance on the gaming machine where the game of chance is a video slot game, a mechanical slot game, a lottery game, a video poker game, a video black jack game, a video card game, a video bingo game, a video keno game and a video pachinko game.

Another aspect of the present invention provides a method of verifying the authenticity of a process temporarily stored in RAM of a gaming machine having a master gaming processor for executing the process. The method may be generally characterized as including: (a) identifying a list of processes scheduled for execution on the gaming machine RAM; (b) selecting one process for verification from the list of processes; (c) identifying a file name and current RAM location of the selected process; (d) at the current RAM location, inspecting the selected process to identify at least a first portion of the process, which first portion of the process is a portion of the process that does not change during execution of the process; (e) identifying one or more gaming software programs stored on one or more file storage devices, which gaming software programs have the same name as the selected process; (f) for each of the one or more identified gaming software programs, inspecting the gaming software programs to determine whether at least the first portion of the process is present; and (g) generating a notification if none of the one or more gaming software programs contains the first portion of the selected process.

In particular embodiments, the gaming software programs may be certified for execution on the gaming machine in one or more gaming jurisdictions by a regulatory entity within each of the gaming jurisdictions. The game of chance may be a video slot game, a mechanical slot game, a lottery game, a video poker game, a video black jack game, a video card game, a video bingo game, a video keno game and a video pachinko game. The method may include: 1) presenting a game of chance on the gaming machine, 2) calling an attendant if none of the one or more gaming software programs contains the first portion of the selected process, 3) shutting down the

gaming machine if none of the one or more gaming software programs contains the first portion of the selected process

Yet another aspect of the present invention provides a method of initializing a gaming system that stores gaming software in RAM on a gaming machine used to present one or more games of chance to a game player. The method may be generally characterized as including: (a) loading a list of gaming software file names from a static memory storage device on the gaming machine; (b) loading a code authenticator program used to compare the list of gaming software file names to names of files stored on a memory storage device on the gaming machine; (c) validating the code authenticator program; (d) comparing the list of gaming software file names with the names of files stored on the memory storage device; (e) when one or more file names on the list of gaming software file names match the names of one or more files stored on the memory storage device, launching the gaming system on the gaming machine.

The method may also include one or more of the following: 1) launching a code comparator program used to compare at least a first portion of a first gaming program temporarily stored in RAM with a first portion of a second gaming software program stored on the memory storage device, 2) when the code authenticator program is not validated, halting the launch of the gaming system on the gaming machine, 3) when one or more file names on the list of gaming software file names does not match the names of one or more files stored on the memory storage device, halting the launch of the gaming system on the gaming machine.

Another aspect of the present invention provides a gaming machine. The gaming machine may be generally characterized as including: 1) a master gaming controller that controls a game of chance played on the gaming machine where the master gaming controller includes: (i) one or more logic devices designed or configured to execute a plurality of gaming software programs used to present the game of chance on the gaming machine and (ii) a RAM that temporarily stores one or more of the plurality of gaming software programs during execution; and 2) gaming logic for comparing a first portion of a first gaming software program as currently stored in the gaming machine RAM with at least a first portion of a second gaming software program. The second gaming software program may be certified for execution on the gaming machine in one or more gaming jurisdictions by a regulatory entity within each of the gaming jurisdictions and may be a substantially identical copy of the first gaming software program. The game of chance is a video slot game, a mechanical slot game, a lottery game, a video poker game, a video black jack game, a video card game, a video bingo game, a video keno game and a video pachinko game.

In particular embodiments, the gaming machine may also include: 1) a file storage device storing the second gaming software program where the file storage device is selected from the group consisting of a hard drive, a CD-ROM drive, a CD-DVD drive and other mass storage devices, 2) gaming logic designed to locate the second gaming software program in a file structure with a plurality of file names and 3) a static memory storage device storing the gaming logic designed to locate the second gaming software program. The static memory storage device may be selected from the group consisting of an EPROM, a flash memory, a non-volatile memory storage device. A list of gaming software file names may also be stored on the static memory storage device where the gaming software files on the list are approved for execution on the gaming machine.

Another aspect of the present invention provides a gaming machine network. The gaming machine network may be generally characterized as including: 1) a plurality of file storage devices storing gaming software programs; 2) a plurality of gaming machines and 3) a network allowing communication between the file storage devices and the plurality of gaming machines. The gaming machines in the game network may be characterized as including: a) a master gaming controller that controls a game of chance played on the gaming machine and b) gaming logic for comparing a first portion of a first gaming software program as currently stored in the gaming machine RAM with at least a first portion of a second gaming software program stored on at least one of the plurality of file storage devices. The master gaming controller in each gaming machine may include (i) one or more logic devices designed or configured to execute a plurality of gaming software programs used to present the game of chance on the gaming machine; and (ii) a RAM that temporarily stores one or more of the plurality of gaming software programs during execution. The network allowing communications between the gaming machines and file storage devices may include the Internet.

Another aspect of the invention pertains to computer program products including a machine-readable medium on which is stored program instructions for implementing any of the methods described above. Any of the methods of this invention may be represented as program instructions and/or data structures, databases, etc. that can be provided on such computer readable media.

These and other features of the present invention will be presented in more detail in the following detailed description of the invention and the associated figures.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1A is block diagram of a gaming machine.

FIGURES 1B and 1C are block diagrams of gaming machines connected to
5 remote storage devices.

FIGURE 2 is a perspective drawing of a gaming machine having a top box and other devices.

FIGURE 3 is a block diagram of a gaming process file structure.

FIGURE 4 is a flow chart depicting a method of verifying the authenticity of a
10 process temporarily stored in RAM.

FIGURE 5 is a flow chart depicting a method of parsing an address space (AS) file.

FIGURE 6 is a flow chart depicting a method of locating authentic process files.

FIGURE 7 is a flow chart depicting a method of initializing an authenticator
15 and code comparator on a gaming machine.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

FIGURE 1A is block diagram of a gaming machine 102 for one embodiment of the present invention. A master gaming controller 101 is used to present one or
20 more games on the gaming machine 102. The master gaming controller 101 executes a number of gaming software programs to operate gaming devices 112 (see FIG. 2) such as coin
hoppers, bill validators, coin acceptors, speakers, printers, lights, displays (e.g. 110) and input mechanisms. One or more displays, such as 110, may be used on the gaming machine. The one or more displays may be mechanical displays (e.g., slot
25 reels), video displays or combinations thereof. The master gaming controller 101 may execute gaming software enabling complex graphical renderings to be presented on one or more displays that may be used as part of a game outcome presentation on the

gaming machine 102. The master gaming controller 101 may also execute gaming software enabling communications with gaming devices located outside of the gaming machine 102, such as player tracking servers and progressive game servers. In some embodiments, communications with devices located outside of the gaming machine may be performed using the main communication board 108 and network connection 125.

In the present invention, for both security and regulatory purposes, gaming software executed on the gaming machine 102 by the master gaming controller 101 is regularly verified by comparing software stored in RAM 106 for execution on the gaming machine 102 with certified copies of the software stored on the gaming machine (e.g. files may be stored on file storage device 114), accessible to the gaming machine via a remote communication connection or combinations thereof. Two gaming software units are used to implement this method: 1) a code comparator and 2) a code authenticator. The code comparator, described in more detail with respect to FIGs. 3, 4 and 5 compares at least some portion of the gaming software scheduled for execution on the gaming machine at a particular time with authenticated gaming software stored in a file storage media accessible to the gaming machine 102. The file storage media may comprise one or more file storage devices, such as 114, located on the gaming machine 102, on other gaming machines, on remote servers or combinations thereof. During operation of the gaming machine, the code comparator frequently checks the gaming software programs being executed by the master gaming controller 101 as the gaming software programs executed by the master gaming controller 101 may vary with time.

The code authenticator, described in more detail with respect to FIGs. 6 and 7 locates on the file storage media an authentic copy of the gaming software being checked by the code comparator. During the boot process for the gaming machine 102 (see FIG. 7), the code authenticator may be loaded from an EPROM such as 104. The master gaming controller 101 executes various gaming software programs using one or more processors such as CPU 103. During execution, a software program may be temporarily loaded into the RAM 106. Depending on the current operational state of the gaming machine, the number types of software programs loaded in the RAM 106 may vary with time. For instance, when a game is presented, particular software programs used to present a complex graphical presentation may be loaded into RAM

106. However, when the gaming machine 102 is idle, these graphical software programs may not be loaded into the RAM.

5 The code comparator and code authenticator execute simultaneously with the execution of the other software programs on the gaming machine. Thus, the gaming machine is designed for “multi-tasking” i.e. the execution of multiple software programs simultaneously. The code comparator and code authenticator processes are most typically used to verify executable code. However, the present invention is not limited to the verification of executable code. It may also be applied to verify any data structures or other information loaded into RAM from mass storage devices used in
10 the presentation of a game on a gaming machine or in any other gaming service provided by the gaming machine.

Details of gaming software programs that may be executed on a gaming machine and an object oriented software architecture for implementing these software programs are described in co-pending U.S. patent application 09/642,192, filed on
15 8/18/00 and entitled “Gaming Machine Virtual Player Tracking and Related Services,” which is incorporated herein in its entirety and for all purposes and co-pending U.S. patent application 09/690,931 filed on 10/17/200 and entitled “High Performance Battery Backed Ram Interface” which is incorporated herein in its entirety and for all purposes.

20 Various gaming software programs, loaded into RAM 106 for execution, may be managed as “processes” by an operating system used on the gaming machine 102. The operating system may also perform process scheduling and memory management. An example of an operating system that may be used with the present invention is the QNX operating system provided by QNX Software Systems, LTD (Kanata, Ontario,
25 Canada).

The code comparator may use information provided by the operating system, such as process information for processes scheduled by the operating system, to select gaming software executables for verification. The QNX operating system provides a list of process that are currently being executed on the gaming machine and
30 information about each process (See FIG. 3). With QNX, the code comparator and code authenticator may be processes scheduled by the operating system.

The present invention is not limited to an operating system such as QNX. The code comparator may be used with other operating systems that provide information about the software programs currently being executed by the operating system and the memory locations of these software units during execution to verify the gaming software programs executing on the gaming machine. For instance, the code comparator may be used with Linux (Redhat, Durham, North Carolina), which is an open source Unix based operating system, or Windows NT or MS Windows 2000 (Microsoft, Redmond, Washington). Windows utilizes a RAM image on the hard drive to create a virtual paging system to manage executable code. The present invention may be applied to verify executable code managed by a virtual paging system. Further, the executable formats and dynamic link libraries between operating systems may vary. The present invention may be applied to different executable formats and link libraries used by a particular operating system and is not limited to the format and libraries of a particular operating system.

The code authenticator searches a file system available to the gaming machine for certified/authentic copies of gaming software programs currently being executed by the gaming machine. The file system may be distributed across one or more file storage devices. The certified/authentic copies of gaming software programs may be certified after a regulatory approval process as described above. The certified/authentic copies of gaming software programs may be stored in a "static" mode (e.g. read-only) on one or more file storage devices located on the gaming machine 102 such as file storage device 114 or EPROM 104. The file storage devices may be a hard-drive, CD-ROM, CD-DVD, static RAM, flash memory, EPROM's, compact flash, smart media, disk-on-chip, removable media (e.g. ZIP drives with ZIP disks, floppies or combinations thereof).

The file system used by the code authenticator may be distributed between file storage devices located on the gaming machine or on remote file storage devices. FIGURES 1B and 1C are block diagrams of gaming machines connected to remote storage devices. In FIG. 1B, gaming machine 102 is connected to two remote file storage devices 116 and 118. The code authenticator may search the two remote file storage devices 116 and 118 as well as local file storage device 114 for gaming software programs that correspond to gaming software programs currently scheduled for execution by the master gaming controller 101. Using a resource sharing system, a

number of gaming software programs may be simultaneously scheduled for execution on the gaming machine at any one time. The resource sharing system, usually embedded in the operating system, develops a sequence order for executing the combination of gaming software programs. When the code authenticator returns a file name and file location (e.g. one of the file storage devices), the code comparator may compare portions of the software program being executed on the gaming machine with a corresponding software program stored one of the file storage devices. The gaming software programs identified by the code authenticator may be in an executable "object" format that includes programming instructions substantially identical to the format of the programming instructions executing on the gaming machine.

In one embodiment a majority of gaming software programs used on the gaming machine may stored on a remote device such as a game server. In FIG. 1C, three gaming machines, 120, 121 and 122 are connected to a game server 124. In this example, the gaming machines 120, 121 and 122 do not include a local file storage device such as a hard drive and gaming executables may be downloaded from the game server 124. The game server may be a repository for game software objects and software for other game services provided on the gaming machine. On each of the gaming machines 120, 121 and 122, the code comparator may compare software being executed by the gaming machine with certified/authentic code stored on the game server 124. One example of a game server that may be used with the present invention is described in co-pending U.S. patent application 09/042,192, filed on 6/16/00, entitled "Using a Gaming Machine as a Server" which is incorporated herein in its entirety and for all purposes. The game server might also be a dedicated computer or a service running on a server with other application programs.

One advantage of the code comparator and code authenticator of the present invention is that gaming software programs executed in a dynamic manner (e.g., different gaming software programs may be continually loaded and unloaded into memory for execution), may be regularly checked to insure the software programs being executed by the gaming machine are certified/authentic programs. The verification process may be used to ensure that approved gaming software is operating on the gaming machine, which may be necessary to satisfy gaming regulatory entities within various gaming jurisdictions where the gaming machine may operate. The

gaming machine may be designed such that when uncertified/authentic programs are detected, an error condition is generated and the gaming machine shuts down. Thus, the present invention enables software architectures and hardware developed for personal computers to be applied to gaming machines.

5 As another advantage, the code comparator and authenticator may also be used to insure "rogue" programs are not operating on the gaming machine. For instance, one method previously used to tamper with a gaming machine might be to introduce a rogue program onto the gaming machine. For example, rogue programs have been used to trigger illegal jackpots on a gaming machine. The code comparator and authenticator may be used to detect these rogue programs and prevent tampering with the gaming machine.

Turning to FIGURE 2, a video gaming machine 2 of the present invention is shown. Machine 2 includes a main cabinet 4, which generally surrounds the machine interior (not shown) and is viewable by users. The main cabinet includes a main door 8 on the front of the machine, which opens to provide access to the interior of the machine. Attached to the main door are player-input switches or buttons 32, a coin acceptor 28, and a bill validator 30, a coin tray 38, and a belly glass 40. Viewable through the main door is a video display monitor 34 and an information panel 36. The display monitor 34 will typically be a cathode ray tube, high resolution flat-panel LCD, or other conventional electronically controlled video monitor. The information panel 36 may be a back-lit, silk screened glass panel with lettering to indicate general game information including, for example, a game denomination (e.g. \$.25 or \$1). The bill validator 30, player-input switches 32, video display monitor 34, and information panel are devices used to play a game on the game machine 2. The devices are controlled by circuitry (See FIG. 1) housed inside the main cabinet 4 of the machine 2. Many possible games, including mechanical slot games, video slot games, video poker, video black jack, video pachinko, video bingo, video keno, video card games, lottery, and other games of chance may be provided with gaming machines of this invention.

30 The gaming machine 2 includes a top box 6, which sits on top of the main cabinet 4. The top box 6 houses a number of devices, which may be used to add features to a game being played on the gaming machine 2, including but not limited

to: a) speakers 10, 12, 14, a ticket printer 18 which prints bar-coded tickets 20, b) a key pad 22 for entering player tracking information such as an identification code, c) a florescent display 16 for displaying player tracking information, d) a card reader 24 for entering a magnetic striped card containing player tracking information or other input devices for entering player tracking information, e) a speaker/microphone for voice commands and voice recognition, f) biometric input devices such as finger printer for identifying a player, g) a video display screen 44 for displaying various types of video content such as player tracking information, machine status, bonus games and primary games and h) a lighted candle that may be used for signaling purposes such as to get the attention of various casino personnel. In some embodiments, some of these gaming devices may also be incorporated into the main cabinet of the gaming machine 2. The ticket printer 18 may be used to print tickets for a cashless ticketing system. Further, the top box 6 may house different or additional devices than shown in the FIGs. 1. For example, the top box may contain a bonus wheel or a back-lit silk screened panel which may be used to add bonus features to the game being played on the gaming machine. As another example, the top box may contain a display for a progressive jackpot offered on the gaming machine. During a game, these devices are controlled and powered, in part, by circuitry (See FIG. 2) housed within the main cabinet 4 of the machine 2.

Understand that gaming machine 2 is but one example from a wide range of gaming machine designs on which the present invention may be implemented. For example, not all suitable gaming machines have top boxes or player tracking features. Further, some gaming machines have two or more game displays – mechanical and/or video. And, some gaming machines are designed for bar tables and have displays that face upwards. As another example, a game may be generated on a host computer and may be displayed on a remote terminal or a remote computer. The remote computer may be connected to the host computer via a network of some type such as the Internet or an intranet. Those of skill in the art will understand that the present invention, as described below, can be deployed on most any gaming machine now available or hereafter developed.

The present invention is not limited to gaming machine and may be applied on other gaming devices executing gaming software from RAM. For example, the gaming devices may include player tracking devices mounted to the gaming machine,

ticket validation systems, hand-held gaming devices and game servers. For example, as described, with respect to FIG. 1, a gaming machine may load gaming software applications from a remote game server in communication with the gaming machine. In this example, the game server and the gaming machine may apply the code comparator and code authenticator processes described in the present invention to verify game software and game data used to provide various gaming services. As another example, a player tracking unit mounted to the gaming machine may be used to provide a plurality of gaming services on the gaming machine. The player tracking unit may include a processor, RAM and mass storage device separate from the gaming machine. The present invention may applied on the player tracking unit to provided verification of gaming software executed on the player tracking unit.

The methods of the present invention may also be applied for remote checks of a gaming device. For example, in one embodiment, a gaming machine may verify the gaming software executing on a player tracking unit connected to the gaming machine. In another example, a game server may remotely verify the gaming software executing on one or more gaming machines in communication with the game server.

Returning to the example of Figure 2, when a user wishes to play the gaming machine 2, he or she inserts cash through the coin acceptor 28 or bill validator 30. Additionally, the bill validator may accept a printed ticket voucher which may be accepted by the bill validator 30 as an indicia of credit when a cashless ticketing system is used. At the start of the game, the player may enter playing tracking information using the card reader 24, the keypad 22, and the florescent display 16. Further, other game preferences of the player playing the game may be read from a card inserted into the card reader. During the game, the player views game information using the video display 34. Other game and prize information may also be displayed in the video display screen 44 located in the top box 6.

During the course of a game, a player may be required to make a number of decisions, which affect the outcome of the game. For example, a player may vary his or her wager on a particular game, select a prize for a particular game selected from a prize server, or make game decisions which affect the outcome of a particular game. The player may make these choices using the player-input switches 32, the video display screen 34 or using some other device which enables a player to input

information into the gaming machine. In some embodiments, the player may be able to access various game services such as concierge services and entertainment content services using the video display screen 34 and one more input devices.

During certain game events, the gaming machine 2 may display visual and auditory effects that can be perceived by the player. These effects add to the excitement of a game, which makes a player more likely to continue playing. Auditory effects include various sounds that are projected by the speakers 10, 12, 14. Visual effects include flashing lights, strobing lights or other patterns displayed from lights on the gaming machine 2 or from lights behind the belly glass 40. After the player has completed a game, the player may receive game tokens from the coin tray 38 or the ticket 20 from the printer 18, which may be used for further games or to redeem a prize. Further, the player may receive a ticket 20 for food, merchandise, or games from the printer 18.

FIGURE 3 is a block diagram of a gaming process file structure 300. As a player utilizes a gaming machine in the manner described above, many different software programs may be executed by the gaming machine. As different gaming software programs are executed by the gaming machine, an operating system running on the gaming machine assign the programs memory location in RAM and then schedule and track the execution of each program as “processes.” The code comparator, which is itself a process, may be used to verify itself and the other processes being executed from RAM.

In one example, every time a process is launched in the operating system, a special directory, such as 310, 315, 320, 325 and 330, is created under the directory “/proc” 305 (e.g. the process directory) in the operating system. The name of this directory is identical to the process ID number (PID) of the process. For instance, process directories corresponding to process ID numbers “1”, “2”, “4049”, “1234” and “6296” are stored under the “/proc” 305 directory. The process directories listed under the “/proc” directory 305 may vary as a function of time as different processes are launched and other process are completed.

In one embodiment, under each PID directory, such as 310, 315, 320, 325 and 330, an address space (AS) file, titled “AS”, may be stored. The AS files, such as 335,

340, 345, 350 and 355 may contains various information about its parent process. Items stored in this file may include, among other things, the command line name used to launch the program and it's location in RAM (e.g. 350) and the names and location in RAM of the shared objects (so) that the process uses (e.g. 352, 354 and
5 356). A shared object is a gaming software program that may be shared by a number of other gaming software programs.

The shared objects used by a process on the gaming machine may vary with time. Thus, the number of shared objects such as 352, 354 and 356 used by a process may vary with time. For instance, a process for a game presentation on a gaming
10 machine may launch various graphical shared objects and audio shared objects during the presentation of a game on the gaming machine and various combinations of these shared objects may be used at various times in the game presentation. For example, a shared object for a bonus game presentation on the gaming machine may only be used when a bonus game is being presented on the gaming machine. Hence, a process for a
15 bonus game presentation may be launched when a bonus game presentation is required and the process may terminate when the bonus game presentation is completed. When the game presentation process uses the bonus game presentation shared object, the launching and the termination of the bonus game presentation shared object may be reflected in the AS file for the game presentation process.

20 The code comparator may use the AS files to determine which game related processes are currently being executed on the gaming machine. The code comparator may also be a process designated in the "/proc" directory 305. Also, in the "/proc" directory there may exist one or more directories that are not representations of process Ids. These include, but are not limited to, SELF, boot 330, ipstats, mount,
25 etc. When parsing the "/proc" directory, these directories are skipped as they do not represent game related code. Once a valid directory is found, e.g., "4049" 320, it is opened and the "AS" file in it may parsed. A detailed method of using the "AS" file as part of a code validation/authentication process is described with respect to FIG. 4.

FIGURE 4 is a flow chart depicting a method 400 of validating the
30 authenticity of a process temporarily stored in RAM on a gaming machine using a code comparator process executed on the gaming machine for one embodiment of the present invention. As described above, the code comparator may be used with other

operating systems which may affect the comparison process. Thus, the following example is provided for illustration purposes only.

5 In 401, the code comparator process is instantiated by the operating system. Various processes may be scheduled for execution on the gaming machine at the same time. Thus, the operating system determines the order in which to execute each process. An execution priority may be assigned to each process. Thus, processes with a higher priority will tend to execute before lower priority processes scheduled to run on the gaming machine.

10 In one embodiment, the code comparator process may be scheduled to run at a low priority where the comparator process may be automatically launched at regular intervals by the operating system. Therefore, during its execution, the code comparator may be preempted by other higher priority processes that may add/remove/reload additional processes. For this reason, the design of the code comparator may include methods to detect when the execution of the code comparator
15 has been preempted and methods to respond to the addition/removal/reloading of processes that may have occurred while the code comparator was preempted.

In other embodiments, the code comparator may not always be a low-level process. During certain states of the gaming machine, the code comparator may be scheduled as a high priority process. For instance, when the code comparator has not
20 been executed over a specific period of time, the priority of the code comparator may be increased until the process is completed. In another example, the code comparator may be launched and complete its tasks without interruption from other processes.

In 405, after the code comparator process has been launched, the comparator process begins to check each process instantiated by the operating system that is listed
25 under the “/proc” directory as described with respect of FIG. 3. It is necessary that the code comparator can open the “/proc” directory. When it can not open the directory, an error is generated as described with respect to FIG. 5. The comparator may check PID directories in a certain range of integer values. PID directories within the range of integer values may correspond to gaming software programs verified by the code
30 comparator while PID directories outside of the integer range may not be verified by the code comparator.

In 410, the code comparator opens the “AS” as described with respect to FIG. 3. When the “AS” file can not be opened, an error condition may be triggered. In 415, when the “AS” file is opened, the code comparator parses process information such as an executable file name corresponding to the process and a temporary memory
5 location of the process in RAM. In addition, the code comparator may parse from the “AS” file the executable file names and temporary memory locations of the processes in RAM for one or more shared objects used by the process. When information from the “AS” file can not be obtained by the code comparator a number of error conditions may be triggered. Further details of 410 and 415 involving opening and
10 parsing the “AS” file are described with respect to FIG. 5.

In 420, when the code comparator process has obtained a file name corresponding to the process in the “AS” file, the location of the file is requested from the code authenticator via an inter process communication (IPC) from the code comparator. IPCs allow processes instantiated by the operating system to share
15 information with one another. When asking the code authenticator for the location(s) of a given file, the full file name and a vector of string pointers, i.e., vector <String *>, are passed. The code authenticator application program interface (API) fills the vector with a list of paths to file locations corresponding to the file name received from code authenticator and returns the vector to the code comparator via an IPC. The
20 list of paths correspond to matching files found on the file storage media (for example, see FIGs. 1A, 1B and 1C) searched by the code authenticator. If no matches are found, the vector returned by the authenticator is empty or may contain an error message. Details of one search method used by the code authenticator is described with respect to FIG. 6.

25 In 425, the code comparator examines the vector returned by the code authenticator. When the vector is empty, the process identified by the code comparator may be considered a rogue process. In 430, an error condition, such as “file not found”, may be reported by the code comparator. The error condition may cause the system manager on the gaming machine to take an action such as shutting
30 down, rebooting, calling an attendant, entering a “safe” mode and combinations thereof.

In 435, operating instructions temporarily stored in RAM corresponding to a process executing on the gaming machine are compared with a certified/authentic operating instructions stored in a file located by the code authenticator. In the operating system for one embodiment of the present invention, files are stored using an Executable and Linking Format (ELF). Details of the ELF format are described as follows and then a comparison by the code comparator of operating instructions for a process stored in RAM with operating instructions stored in a corresponding ELF file are described.

There are three ELF file types: 1) executable, 2) relocatable and 3) shared object. Of these three, only the executable and shared object formats, which may be executed by the operating system, are used by the code comparator. There are five different sections that may appear in any given ELF file including a) an ELF header, b) a program header table, c) section header table, d) ELF sections and e) ELF segments. The different sections of the ELF file are described below.

The first section of an ELF file is always the ELF Header. It is the only section that has a fixed position and is guaranteed to be present. The ELF header has three tasks: 1) it details the type of file, target architecture, and ELF version, 2) it contains the location within the file of the program headers, section headers, and string tables as well as their size and 3) it contains the location of the first executable instruction.

The Program Header Table is an array of structures that can each describe either a segment in the file or provide information regarding creating an executable process image. Both the size of each entry in the program header table and the number of entries reside in the ELF header. Every entry in the program header table includes a type, a file offset, a physical and virtual addresses, a file size, a memory image size and a segment alignment. Like the program header table, the section header table contains an array of structures. Each entry in the section header table contains a name, a type, a memory image starting address, a file offset, a size an alignment and a section purpose. For every section in the file, a separate entry exists in the section header table.

Nine different ELF section types exist. These consist of executable, data, dynamic linking information, debugging data, symbol tables, relocation information, comments, string tables and notes. Some of these types are loaded into the process image, some provide information regarding the building of the process image, and some are used when linking object files. There are three categories of ELF segments: 1) text, 2) data and 3) dynamic. The text segment groups executable code, the data segment groups program data, and the dynamic segment groups information relevant to dynamic loading. Each ELF segment consists of one or more sections and provide a method for grouping related ELF sections. When a program is executed, the operating system interprets and loads the ELF segments to create a process image. If the ELF file is a shared object file, the operating system uses the segments to create the shared memory resource.

In 435, the comparison process may include first verifying the ELF header and then verifying the program blocks. When a program is temporarily loaded in RAM as a process, only the program blocks that are marked as loadable and executable in the ELF file will exist in RAM and, therefore, are the only ones verified.

To validate a process loaded in RAM, the code comparator opens a file on the storage device where the file is located. The code comparator begins with the first file in the vector of file paths sent to the code comparator by the code authenticator. In 415, the RAM address of the loaded process is obtained from "AS" when the "AS" file is parsed. The RAM address marks the start of the loaded ELF header. The loaded ELF header is verified against the corresponding ELF header from the file on the storage device. Since the size of the ELF header is fixed, this comparison is a straight forward byte comparison. If the ELF header matches, the program blocks are then checked.

The code comparator may consider two things when comparing ELF program blocks. First, what program blocks were loadable and/or executable and second, where do each of the program blocks reside in RAM. The number of program headers resides in the ELF header. Each of these headers, in turn, contains the offset to the code block that they represent as well as whether or not it is loadable or executable.

The starting address for where, in RAM, the code exists, resides in the "AS" file. This is the same for the file except that the starting address of the file pointer is used to determine the start of the program. All executable/loadable program blocks in RAM are compared against the file stored on the storage media. Data blocks which
5 may vary as the program is executed are not usually checked. However, in some programs, "fixed" or static data blocks may be checked by the code comparator. In one embodiment, when all blocks check out, the comparison is deemed successful. In another embodiment, only a portion of the program blocks may be checked by the code comparator. To decrease the time the comparison process takes, partial or
10 random section portions of code may be compared. In one embodiment, a bit-wise comparison method is used to compare code. However, the method is not limited to a bit-wise comparison other comparison methods may be used or combinations of comparison methods may be used.

During the file comparison process, a mismatch may result from several
15 different conditions including but not limited to the conditions described as follows. First, it is possible that the code comparator was pre-empted and that the process that is currently being verified was terminated. Second, it is also possible that the RAM contents or file contents for the process in question may have been corrupted. Third, the file being compared could have the same name as the file used to launch to
20 process but not actually be the same file. This condition may occur when the code authenticator returns a vector with multiple file paths corresponding to the file name sent to the code authenticator by the code comparator. Fourth, the process executing in RAM may have been altered in some manner in an attempt to tamper with the gaming machine.

25 In 440, the code comparator checks the status of the RAM and file compare process. In 445, when the compare is accepted (the conditions for accepting the compare may be varied), the code comparator begins to check any shared objects for the process obtained from the "AS" file. When the process does not use shared objects, the code comparator continues to the next PID directory in 405. When the
30 process is using one or more shared objects, the code comparator sends a request to the code authenticator to find file locations corresponding to the file name for the shared object in 420.

In 442, when a mismatch occurs, to determine whether the process has terminated, the "AS" file for the process is re-parsed and the newly obtained contents are compared against the original contents obtained initially. When the "AS" file is no longer accessible, the process was terminated during the compare process and the comparison is aborted and an error condition is not generated. When the "AS" file can be re-parsed but the file name stored within the "AS" file has changed, then the original process may be terminated and a new process may have been started with the same process identification number (PID). In this case, the comparison process is aborted and error condition is not generated.

In 445, when the newly obtained contents from the "AS" file match the original contents of the "AS" file in 442, the comparison process continues with the next file from the matching file list in the vector that was obtained via the code authenticator process. When the code comparator reaches the end of this vector list without matching the process, a rogue process may be running and an error condition is reported in 450 to the system manager. In 440, when a comparison fails because of a RAM and/or file corruption, the comparator may check whether the process has terminated in 442 and continue to the next the file in the authenticator file list in 445. Once the end of the authenticator file list is reached, the comparator will declare a rogue process and report the error in 450.

FIGURE 5 is a flow chart depicting a method of parsing an address space (AS) file as described with respect to 410 and 415 in FIG. 4. The method is presented for illustrated purposes as it is specific to the QNX operating system. A similar method may be developed for different operating systems such as Linux or Windows NT. In 500, the code comparator attempts to open the process directory ("/proc" as described with reference to FIG. 3), which is provided by the operating system and contains a list of all the processes currently scheduled for execution. In 505, when the process directory can not be opened, an error is sent by the code comparator to the system manager indicating the process directory can not be opened. In one example, the process directory as well as other directories below the process directory may be inaccessible because an access privilege has been set on the directory that prevents access by the code comparator. Access privileges for directories are well known in UNIX based operating systems such as QNX.

In 510, when the process directory can be opened, the code comparator selects the next directory in the list of PID directories under the process directory. The PID directories are listed as integers. The code comparator, which may be repeatedly pre-empted by other process while performing the code comparison, stores which integer
5 PID it is currently comparing and then proceeds to the next closet integer after the compare on the current process is completed. In 515, the code comparator compares the selected integer PID number with a range of integers. Not all processes are necessarily compared by the code comparator. In general, only processes within a particular numerical range corresponding to gaming software that has been certified
10 are verified by the code comparator. When the PID directory number does not fall within the range of integers checked by the code comparator or the PID directory has a text name, such as boot, the code comparator proceeds to the next PID directory in the process directory in 510.

When the PID directory is within the integer range of processes which the
15 code comparator checks, in 520, the code comparator attempts to open the PID directory. In 521, when the PID directory can not be opened, the comparator determines whether the process was terminated by the operating system. When the process was terminated by the operating system, the code comparator moves to the next directory in the process directory in 510. In 522, when the PID directory can not
20 be opened and the process was not terminated by the operating system, an error message is posted to the operating system. A way of tampering with the gaming machine may be to generate a process that can not be checked by the code comparator.

In 525, when the PID directory can be opened, the code comparator attempts
25 to open the Address Space (AS) file as described with reference to FIG. 2. The "AS" file may contain a process memory address location, a process executable file name, shared object memory address locations used by the process and shared object executable file names corresponding to the shared objects. In 540, the code comparator attempts to read the "AS" file. In 550, when the file is readable, the code
30 comparator continues with the comparison process according to 420 in FIG. 4.

In 540 when the code comparator can not get information from the "AS" file, the code comparator checks for the "Error for Search (ESRCH)" error condition in

545. The error code ESRCH is returned when the requested file does not exist and indicates that the process the code comparator was trying to access was removed. When the code comparator detects this error code, the error is ignored and the code comparator continues to the next PID directory in 510. In 555, when an ERSCH error condition is not detected, an error message is sent to the system manager indicating the “AS” file can not be parsed. The “AS” may not be parsable for a number of reasons. For instance, the data in the “AS” may have been corrupted in some manner that prevents the code comparator from reading the file.

10 In 525 when the “AS” can not be opened, only one error code, “Error No Entry (ENOENT)” is tolerated. The ENOENT error code is returned when the requested file does not exist. It indicates that the process the code comparator was trying to access was removed by the operating system. In 530, the code comparator checks for the ENOENT code. When an ENOENT error code has been generated, the code is ignored and the code comparator moves on to the next PID directory in 510.

15 The ENOENT code may have been generated because the code comparator was preempted during its operation by the execution of one or more higher priority processes. While the higher priority processes were being executed, the process that the code comparator was checking may have been terminated. When any other error code is detected by the code comparator, in 535 an error message is sent to the

20 operating system indicating that the “AS” can not be opened. For instance, the “AS” file may exist but the code comparator may not have the access privilege to open the file which would generate an error condition other than ENOENT and hence an error condition in 535.

FIGURE 6 is a flow chart depicting a method of locating authentic process files. In 420, as described above, the comparator sends a file name request via an interprocess communication to the code authenticator. In 605, via the code authenticator application program interface, the code authenticator receives a file name. The code authenticator searches through a list of file names where each file name corresponds to certified executable gaming software program. The certified gaming software programs may be stored on storage media, i.e. one or more file storage devices, located within the gaming machine, located outside of the gaming machine or combinations thereof. A portion of the certified executable gaming software programs may have been approved by a gaming regulatory agency in a

gaming jurisdiction for use on gaming machines in the gaming jurisdiction. In cases where a gaming jurisdiction does not require certification of a particular software program, the gaming software program may also be certified as authentic by the gaming manufacturer for security reasons. Further details of code authenticator application may be found in co-pending U.S. Application no. 09/643,388, filed on August 21, 2000, by LeMay, et al., "Method and Apparatus for Software Authentication" which is incorporated in its entirety and for all purposes.

In 610, the code authenticator determines whether it has reached an end of the list of certified file names. When the code authenticator has not reached the end of the list, in 615, the code authenticator gets the next file name of the list. In 620, when the name from the list matches the name received from the code comparator, the path to the file, which may be the location of the file in a file structure stored on a file storage device, is added to a list of matched files detected by the code comparator.

The list of matched files is stored in a vector which may contain zero files when no files have been matched to a plurality of files when multiple matches have been detected by the code comparator. In the case where multiple matches have been detected, the multiple files may reside on a common file storage device or the multiple files may reside on different file storage devices. In 620, when a match is not detected, the code authenticator checks the next file entity on the list for a match. In 630, after the entire list of certified file names has been searched, the authenticator sends a vector, which may be empty, containing a list of matches detected by the code authenticator, to the code comparator via an IPC.

FIGURE 7 is a flow chart depicting a method 800 of initializing an authenticator and code comparator on a gaming machine. In 805, the code authenticator is loaded by the BIOS from an EPROM (see FIGs. 1A-1C). The code authenticator may be stored on an EPROM for security and gaming approval reasons. The EPROM storing the code authenticator can be submitted for approval to a gaming jurisdiction. Once the EPROM has been approved, as was previously described, a unique signature may be generated for the EPROM. The unique signature may be checked when the EPROM is installed on the gaming machine in the local gaming jurisdiction. Since software stored on the EPROM is generally difficult to alter, the use of the EPROM may also prevent tampering with the gaming machine.

In 810, after the code authenticator has been loaded from the EPROM, the code authenticator may validate itself. For instance, a CRC may be performed on the authenticator software to obtain a CRC value. The CRC value may be compared with a certified CRC value stored at some location on the gaming machine. In 812, the
5 validation check is performed. When the authenticator is not valid, the initialization of the gaming machine is halted in 835 and the gaming machine may be shutdown or placed in a safe mode. In 815, the code authenticator may compare a list of certified software programs stored in the EPROM with a list of software programs available on the gaming machine. As an example, the EPROM may contain about 1 Megabyte of
10 memory available for storage purposes but is not limited to this amount. The code authenticator may also perform other files system checks.

In 817, file system has not been validated, the launch of the gaming machine is halted and the gaming machine may be shutdown or placed in a safe mode in 835. In 817, when the file system has been validated, the system manager is launched in 820.
15 In 825 and 830, the system manager launches the game manger and the code comparator. Once the code comparator is launched, it continually runs in the background preferably as a task in a “multi-tasking system.”

Although the foregoing invention has been described in some detail for purposes of clarity of understanding, it will be apparent that certain changes and
20 modifications may be practiced within the scope of the appended claims. For instance, while the gaming machines of this invention have been depicted as having top box mounted on top of the main gaming machine cabinet, the use of gaming devices in accordance with this invention is not so limited. For example, gaming machine may be provided without a top box.

25